Technische Universität Berlin

FACHBEREICH 3
MATHEMATIK

SCHEDULING–LPS BEAR PROBABILITIES

RANDOMIZED APPROXIMATIONS FOR MIN–SUM CRITERIA

(*Extended Abstract*)

ANDREAS S. SCHULZ MARTIN SKUTELLA

No. 533/1996

Scheduling–LPs Bear Probabilities*

Randomized Approximations for Min–Sum Criteria

(*Extended Abstract*)

Andreas S. Schulz** Martin Skutella**

November 1996; revised March 1997

Abstract

In this paper, we provide a new class of randomized approximation algorithms for scheduling problems by directly interpreting solutions to so-called time-indexed LPs as probabilities. The most general model we consider is scheduling unrelated parallel machines with release dates (or even network scheduling) so as to minimize the average weighted completion time. The crucial idea for these multiple machine problems is not to use standard list scheduling but rather to assign jobs randomly to machines (with probabilities taken from an optimal LP solution) and to perform list scheduling on each of them.

For the general model, we give a $(2+\varepsilon)$–approximation algorithm. The best previously known approximation algorithm has a performance guarantee of $16/3$ [HSW96]. Moreover, our algorithm also improves upon the best previously known approximation algorithms for the special case of identical parallel machine scheduling (performance guarantee $(2.89+\varepsilon)$ in general [CPS+96] and 2.85 for the average completion time [CMNS97], respectively). A perhaps surprising implication for identical parallel machines is that jobs are randomly assigned to machines, in which each machine is equally likely. In addition, in this case the algorithm has running time $O(n \log n)$ and performance guarantee 2.

For minimizing the average weighted completion time on a single machine under release dates, a refined version of our algorithm produces in $O(n \log n)$ time a schedule that is expected to be within a factor of 1.6853 of the optimum. An appropriately adapted version is a $4/3$–approximation algorithm for preemptive single machine scheduling to minimize the average weighted completion time subject to release dates. This improves upon a 1.466–approximation algorithm due to Goemans, Wein, and Williamson [GWW96].

Finally, the results for identical parallel machine as well as single machine scheduling apply to both the off-line and the on-line settings with no difference in performance guarantees. In the on-line setting, we are scheduling jobs that continually arrive to be processed and, for each time t, we must construct the schedule until time t without any knowledge of the jobs that will arrive afterwards.

1 Introduction

It is by now well-known that randomization can help in the design of algorithms, cf., e. g., [MR95, MNR96]. One way of guiding randomness is the use of linear programs (LPs). In this paper, we give LP-based approximation algorithms for problems which are particularly well-known for the difficulties to obtain good

*NOTE ADDED IN MAY 1997: The work on $1|r_j|\sum w_j C_j$ presented in Section 3 of this preprint will be combined with the work of Goemans, Queyranne, and Wang [Wan96, Goe96a] to a joint journal paper [GQS+97]. The multiple machine results will appear in [SS97b].

**Technische Universität Berlin, Fachbereich Mathematik, MA 6–1, Straße des 17. Juni 136, 10623 Berlin, Germany, E–mail {schulz,skutella}@math.tu–berlin.de

lower bounds: machine (or processor) scheduling problems. Because of the random choices involved, our algorithms are rather randomized approximation algorithms. A randomized ρ–approximation algorithm is a polynomial-time algorithm that produces a feasible solution whose expected value is within a factor of ρ of the optimum; ρ is also called the expected performance guarantee of the algorithm. Actually, we always compare the output of an algorithm with a lower bound given by an optimum solution to a certain LP relaxation. Hence, at the same time we obtain an analysis of the quality of the respective LP. All our off-line algorithms can be derandomized with no difference in performance guarantee, but at the cost of increased (but still polynomial) running times. For reasons of brevity, we always omit the technical details of derandomization.

We consider the following model. We are given a set J of n jobs (or tasks) and m unrelated parallel machines. Each job j has a positive integral processing requirement p_{ij} which depends on the machine i job j will be processed on. Each job j must be processed for the respective amount of time on one of the m machines, and may be assigned to any of them. Every machine can process at most one job at a time. In *preemptive* schedules, a job may repeatedly be interrupted and continued later on another (or the same) machine. In *nonpreemptive* schedules, a job must be processed in an uninterrupted fashion. Each job j has an integral release date $r_j \geqslant 0$ before which it cannot be processed. We denote the completion time of job j in a schedule by C_j, and for any fixed $\alpha \in (0,1]$, the α–point $C_j(\alpha)$ of job j is the first moment in time at which an α–fraction of job j has been completed; α–points were first used in the context of approximation by [HSW96]. We seek to minimize the total weighted completion time: a weight $w_j \geqslant 0$ is associated with each job j and the goal is to minimize $\sum_{j\in J} w_j C_j$. In scheduling, it is quite convenient to refer to the respective problems using the standard classification scheme of Graham et al. [GLLRK79]. The nonpreemptive problem $R\,|\,r_j\,|\sum w_j C_j$, just described, is strongly NP-hard.

Scheduling to minimize the total weighted completion time (or, equivalently, the average weighted completion time) has recently achieved a great deal of attention, partly because of its importance as a fundamental problem in scheduling, and also because of new applications, for instance, in compiler optimization [CJM+96] or in parallel computing [CM96]. In the last two years, there has been significant progress in the design of approximation algorithms for this kind of problems which led to the development of the first constant worst-case bounds in a number of settings. This progress essentially follows on the one hand from the use of preemptive schedules to construct nonpreemptive ones [PSW95, CPS+96, CMNS97, Goe97]. On the other hand, one solves a linear programming relaxation and then a schedule is constructed simply by list scheduling in a natural order dictated by the LP solution [PSW95, HSW96, Sch96, HSSW96, MSS96, Goe97, CS97].

In this paper, we introduce a different idea: random assignments of jobs to machines. To be more precise, we exploit a new LP relaxation for the problem $R\,|\,r_j\,|\sum w_j C_j$, and we then show that a certain variant of randomized rounding leads to a $(2+\varepsilon)$–approximation algorithm, for any $\varepsilon > 0$. At the same moment, the corresponding LP is a $(2+\varepsilon)$–relaxation, i. e., the true optimum is always within a factor of $(2+\varepsilon)$ of the optimal value of the LP relaxation; and this is tight. Our algorithm improves upon a $16/3$–approximation algorithm of Hall, Shmoys, and Wein [HSW96] that is also based on time-indexed variables which have a different meaning, however. In contrast to their approach, our algorithm does not rely on Shmoys and Tardos' rounding technique for the generalized assignment problem [ST93]. We rather exploit the LP by interpreting LP values as probabilities with which jobs are assigned to machines. For an introduction to and the application of randomized rounding to other combinatorial optimization problems, the reader is referred to [RT87, MNR96].

For the special case of identical parallel machines, i. e., for each job j and all machines i we have $p_{ij} = p_j$, Chakrabarti, Phillips, Schulz, Shmoys, Stein, and Wein [CPS+96] obtained a $(2.89+\varepsilon)$–approximation by

refining an on-line greedy framework of Hall et al. [HSW96]. The former best known LP-based algorithm, however, relies on an LP relaxation solely in completion time variables which is weaker than the one we propose. It has performance guarantee $(4-\frac{1}{m})$ (see [HSSW96] for the details). For the LP we use here, an optimum solution can greedily be obtained by a certain preemptive schedule on a virtual single machine which is m times as fast as any of the original machines. The idea of using a preemptive relaxation on such a virtual machine was introduced before by Chekuri, Motwani, Natarajan, and Stein [CMNS97]. They show that any preemptive schedule on this machine can be converted into a nonpreemptive schedule on the m identical parallel machines such that, for each job j, its completion time in the nonpreemptive schedule is at most $(3-\frac{1}{m})$ times its preemptive completion time. For the average completion time, they refine this to a 2.85–approximation algorithm for $\mathrm{P}\,|\,r_j\,|\,\sum C_j$. For $\mathrm{P}\,|\,r_j\,|\,\sum w_jC_j$, the algorithm we propose delivers in time $\mathrm{O}(n\log n)$ a solution that is expected to be within a factor of 2 of the optimum.

The first constant-factor approximation algorithm to minimize the average weighted completion time on a single machine subject to release dates is due to Phillips, Stein, and Wein [PSW95]. They showed that, given a preemptive schedule, list scheduling in nondecreasing order of the preemptive completion times gives a nonpreemptive schedule such that, for each job, the nonpreemptive completion time is not worse than twice its preemptive one. This job-by-job bound immediately implies a 2–approximation for $1\,|\,r_j\,|\,\sum C_j$ since the corresponding preemptive problem is solvable in polynomial time [Bak74]. Motivated by this idea, Hall, Shmoys, and Wein [HSW96] and then in turn, inspired by the work of Hall et al., Schulz [Sch96] used various LP relaxations to schedule accordingly to nondecreasing "LP completion times". This led to 4– and 3–approximations for $1\,|\,r_j\,|\,\sum w_jC_j$, respectively. Recently, and independently from each other, Chekuri et al. [CMNS97] and Goemans [Goe97] have taken up the ideas of converting preemptive schedules into nonpreemptive ones and list scheduling in order of α–points (as introduced in [HSW96]), and enriched these by the use of randomness. Consider any preemptive schedule with completion times C_j, for job j. Chekuri et al. show that if α is selected at random in $(0,1]$ with density function $e^{\alpha}/(e-1)$, then the expected completion time of job j in the schedule produced by sequencing the jobs in nondecreasing order of α–points is at most $\frac{e}{e-1}C_j$. Again, this immediately leads to an $\frac{e}{e-1}$–approximation algorithm for $1\,|\,r_j\,|\,\sum C_j$. In contrast, Goemans uses a specially structured preemptive schedule (which is the solution to an LP in time-indexed variables), selects α uniformly, and shows that the total weighted completion time of the resulting schedule is expected to be within a factor of 2 of the optimum LP value.

It turns out, however, that Goemans' randomized scheduling by α–points is in a sense too restrictive. It introduces a coupling of jobs which is not present in the LP solution. Our results for unrelated parallel machines imply that the use of individual, randomly chosen α_j's for each job j is, in this sense, closer to the LP, admits a simpler analysis of Goemans' accordingly modified algorithm, and, perhaps most important, leads to an improved analysis which also gives job-by-job bounds. Actually, we show that the expected completion time of every job is not worse than 1.6853 times its corresponding LP completion time. Parallel to and independently from our work, but partly stimulating the authors and partly stimulated by the authors through mutual performance guarantee announcements, both Goemans and Wang improved Goemans' 2–approximation algorithm for $1\,|\,r_j\,|\,\sum w_jC_j$. Wang [Wan96] derived a 1.7735–approximation algorithm which was then improved by Goemans [Goe96a] to 1.745. Finally, during the last Symposium on Discrete Algorithms (SODA'97), Goemans announced that one can combine Wang's algorithm with an earlier version of our algorithm (which had performance guarantee 1.693) to a 1.685...–approximation algorithm. This announcement then inspired us to directly improve our earlier analysis to get a 1.6853–approximation algorithm.

The paper is organized as follows. In Section 2, we start with the discussion of our main result: the 2–approximation algorithm in the general context of unrelated parallel machine scheduling. In the next section,

we point out its simplification for single machine scheduling which especially admits an improved performance guarantee by a more sophisticated use of randomness. In addition, we also show in Section 3 that this approach yields a $4/3$–approximation algorithm for the preemptive variant $1|r_j, pmtn|\sum w_jC_j$. Finally, in Section 4 we combine the more general point of view from unrelated parallel machines with the insights from single machine scheduling to give a combinatorial 2–approximation algorithm for $\mathrm{P}|r_j|\sum w_jC_j$. Some illustrations and technical details which, for reasons of brevity, we have to omit from the extended abstract itself can be found in the appendix.

2 Unrelated Parallel Machine Scheduling with Release Dates

In this section, we consider the problem $\mathrm{R}|r_j|\sum w_jC_j$. As in [HSW96], we will even discuss a slightly more general problem in which the release date of job j may also depend on the machine. The release date of job j on machine i is thus denoted by r_{ij}. Machine-dependent release dates are relevant to model network scheduling in which parallel machines are connected by a network, each job is located at one given machine at time 0, and cannot be started on another machine until sufficient time elapses to allow the job to be transmitted to its new machine. This model has been introduced in [DLLX90, AKP92].

The problem $\mathrm{R}|r_{ij}|\sum w_jC_j$ is well-known to be strongly NP-hard, even for the case of a single machine [LRKB77]. In order to design an approximation algorithm we introduce an LP relaxation whose optimum value serves as a surrogate for the true optimum in our estimations.

Let therefore $T = \max_{i,j} r_{ij} + \sum_{j\in J} \max_i p_{ij} - 1$ be the time horizon, and introduce for every job $j \in J$, every machine $i = 1,\dots,m$, and every point $t = 0,\dots,T$ in time a variable y_{ijt} which represents the time job j is processed on machine i within the time interval $(t,t+1]$. Equivalently, one can say that a y_{ijt}/p_{ij}–fraction of job j is being processed on machine i within the time interval $(t,t+1]$. The LP is as follows:

$$
\begin{aligned}
& & \text{minimize} \quad & \sum_{j\in J} w_jC_j & & \\
(LP_R) & & \text{subject to} \quad & \sum_{i=1}^{m}\sum_{t=r_{ij}}^{T} \frac{y_{ijt}}{p_{ij}} = 1 \quad \text{for all } j\in J & & (1)\\
& & & \sum_{j\in J} y_{ijt} \leqslant 1 \quad \text{for } i = 1,\dots,m \text{ and } t = 0,\dots,T & & (2)\\
& & & C_j = \sum_{i=1}^{m}\sum_{t=0}^{T}\Big(\frac{y_{ijt}}{p_{ij}}\big(t+\tfrac{1}{2}\big) + \tfrac{1}{2}y_{ijt}\Big) \quad \text{for all } j\in J & & (3)\\
& & & y_{ijt} = 0 \quad \text{for } i = 1,\dots,m,\ j\in J,\ t = 0,\dots,r_{ij}-1 & & (4)\\
& & & y_{ijt} \geqslant 0 \quad \text{for } i = 1,\dots,m,\ j\in J,\ t = r_{ij},\dots,T & & (5)
\end{aligned}
$$

Equations (1) ensure that the whole processing requirement of every job is satisfied. The machine capacity constraints (2) simply express that each machine can process at most one job at a time. Consider an arbitrary feasible schedule, where job j is being continuously processed between time $C_j - p_{hj}$ and C_j on machine h. Then the expression for C_j in (3) corresponds to the real completion time, if we assign the values to the LP variables y_{ijt} as defined above, i. e., $y_{ijt} = 1$ if $i = h$ and $t \in \{C_j - p_{hj},\dots,C_j - 1\}$, and $y_{ijt} = 0$ otherwise. Hence, (LP_R) is a relaxation of the scheduling problem under consideration.

The following algorithm takes an optimum LP solution, and then constructs a feasible schedule by using a kind of randomized rounding.

Algorithm R

1) Compute an optimum solution y to (LP_R).
2) Assign job j to a machine–time pair (i,t) at random with probability y_{ijt}/p_{ij}; then draw t_j from the chosen time interval $(t,t+1]$ at random with uniform distribution.
3) Schedule on each machine i the jobs that were assigned to it nonpreemptively as early as possible in nondecreasing order of t_j.

Note that all the random assignments need to be performed independently from each other (at least pairwise). The following lemma illuminates the intuition underlying Algorithm R by relating the implications of Step 2 to the solution y of (LP_R).

Lemma 2.1. *Let y be the optimum solution to (LP_R) which is computed in Step* 1 *of Algorithm* R*. Then, for each job $j \in J$ the following holds:*

a) The expected value of t_j equals $\sum_{i=1}^{m}\sum_{t=0}^{T}\frac{y_{ijt}}{p_{ij}}(t+\frac{1}{2})$.

b) The expected processing time of j in the schedule constructed by Algorithm R *equals $\sum_{i=1}^{m}\sum_{t=0}^{T} y_{ijt}$.*

Proof. First, we fix a machine–time pair (i,t) job j has been assigned to. Then, the expected processing time of j under these conditions is p_{ij}. Moreover, the conditional expectation of t_j is $t+\frac{1}{2}$. By adding these conditional expectations over all machines and time intervals, weighted by the corresponding probabilities y_{ijt}/p_{ij}, we get the claimed results. □

Note that the lemma remains true if we start with an arbitrary, not necessarily optimal solution y to (LP_R) in Step 1 of Algorithm R. This also holds for the following theorem. The optimality of the LP solution is only needed to get a lower bound on the value of an optimal schedule.

Theorem 2.2. *The expected completion time of each job j in the schedule constructed by Algorithm* R *is at most $2\cdot C_j^{LP}$, where C_j^{LP} is the LP completion time (defined by* (3)*) of the optimum solution y we started with in Step* 1*.*

Proof. We consider an arbitrary, but fixed job $j \in J$. To analyze the expected completion time of job j, we first also consider a fixed assignment of j to a machine–time pair (i,t). Then, the expected starting time of job j under these conditions precisely is the conditional expected idle time plus the conditional expected amount of processing of jobs that come before j on machine i.

Observe that there is no idle time on machine i between the maximum release date of jobs on machine i which start no later than j and the starting time of job j. It follows from the ordering of jobs and constraints (4) that this maximum release date and therefore the idle time of machine i before the starting time of j is bounded from above by t.

On the other hand, we get the following bound on the conditional expected processing time on machine i before the start of j:

$$\sum_{k\neq j} p_{ik}\cdot \Pr(k \text{ on } i \text{ before } j) \leqslant \sum_{k\neq j} p_{ik}\cdot \sum_{\ell=0}^{t}\frac{y_{ik\ell}}{p_{ik}} \leqslant t+1 \ .$$

The last inequality follows from the machine capacity constraints (2). Putting the observations together we get an upper bound of $2\cdot(t+\frac{1}{2})$ for the expected starting time of j. Unconditioning the expectation together with Lemma 2.1 b) and (3) yields the theorem. □

Theorem 2.2 implies a performance guarantee of 2 for Algorithm R. We also have shown that (LP_R) is a 2–relaxation. Moreover, even for the case of identical parallel machines without release dates there are instances for which this bound is asymptotically attained, see Section 4. Thus our analysis is tight.

Unfortunately, (LP_R) has exponentially many variables. Consequently, Algorithm R only is a pseudo-polynomial-time algorithm. However, we can overcome this drawback by introducing new variables which are not associated with time intervals of length 1, but rather with intervals of geometrically increasing size. The idea of using interval-indexed variables to get polynomial-time solvable LP relaxations was introduced earlier by Hall, Shmoys, and Wein [HSW96]. In order to get polynomial-time approximation algorithms in this way, we have to pay for with a slightly worse performance guarantee. For any constant $\varepsilon > 0$, we get an approximation algorithm with performance guarantee $2+\varepsilon$ for $\mathrm{R}|r_{ij}|\sum w_jC_j$. The rather technical details can be found in Appendix A.

It is shown in [SS97a] that in the absence of (non-trivial) release dates, the use of a slightly stronger LP relaxation improves the performance guarantee of Algorithm R to $3/2$. Independently, this has also been observed by Fabián A. Chudak (communicated to us by David B. Shmoys, March 1997) after reading a preliminary version of the paper on hand which only contained the 2–approximation for $\mathrm{R}|r_{ij}|\sum w_jC_j$.

Remark 2.3. The reader might wonder whether the seemingly artificial random choice of the t_j's in Algorithm R is really necessary. Indeed, it is not, which also implies that we could work with a discrete probability space: From the proof of Theorem 2.2 one can see that we could simply set $t_j = t$ if job j is assigned to a machine-time pair (i,t) — without loosing the performance guarantee of 2. Ties are simply broken (as before) by preferring jobs with smaller indices, or even randomly. We mainly chose this presentation for the sake of better comparison with related work in the special case of single machine scheduling; this will become apparent soon.

3 Single Machine Scheduling with Release Dates

We now consider the case of a single machine. The processing times and release dates are thus denoted by p_j resp. r_j. There are several good reasons to investigate this special case in its own rights.

- There is a purely combinatorial algorithm to solve the LP relaxation. Therefore, the resulting randomized approximation algorithm has running time $\mathrm{O}(n\log n)$.
- The previous use of randomness obtains another interpretation in terms of scheduling by α–points (and vice versa).
- A more sophisticated use of randomness leads to an improved performance guarantee of 1.6853.
- Further modifications yield a randomized $4/3$–approximation algorithm for the preemptive variant $1|r_j, pmtn|\sum w_jC_j$. Its running time also is $\mathrm{O}(n\log n)$.
- Both algorithms can easily be turned into randomized on-line algorithms, with no difference in performance guarantees.

We first focus on nonpreemptive solutions and give an approximation algorithm that converts a preemptive schedule into a nonpreemptive one. In contrast to Phillips et al. [PSW95] and Chekuri et al. [CMNS97], however, we do not consider arbitrary preemptive schedules, but, as in [Goe97], rather a very structured one. We then show that this algorithm is nonetheless a special case of Algorithm R.

Algorithm 1

1) Construct a preemptive schedule by scheduling at any point in time among the available jobs the one with largest w_j/p_j ratio. Let C_j be the completion time of job j in this preemptive schedule.

2) Independently for each job $j \in J$, draw α_j randomly from $(0,1]$.
3) Schedule the jobs nonpreemptively as early as possible in nondecreasing order of $C_j(\alpha_j)$.

For Step 1 we should add that in case of ties we always prefer the job with smaller index. Notice that whenever a job is released, the job being processed (if any) will be preempted if the released job has a higher w_j/p_j ratio (or the same, but a smaller index).

In the analysis of Algorithm 1, we will again use (LP_R). We denote the variables by y_{jt} corresponding to jobs j and time intervals $(t,t+1]$. Their interpretation is analogous to the one in Section 2. This leads to the following simplified LP that was introduced by Dyer and Wolsey [DW90]:

$$
\begin{aligned}
& & \text{minimize} \quad & \sum_{j\in J} w_j C_j & & \\
(LP_1) \qquad & & \text{subject to} \quad & \sum_{t=r_j}^{T} y_{jt} = p_j \quad \text{for all } j \in J & & (6)\\
& & & \sum_{j\in J} y_{jt} \leqslant 1 \quad \text{for } t = 0,\dots,T & & (7)\\
& & & C_j = \frac{p_j}{2} + \frac{1}{p_j}\sum_{t=0}^{T} y_{jt}\left(t+\tfrac{1}{2}\right) \quad \text{for all } j \in J & & (8)\\
& & & y_{jt} = 0 \quad \text{for all } j \in J \text{ and } t = 0,\dots,r_j-1 & & (9)\\
& & & y_{jt} \geqslant 0 \quad \text{for all } j \in J \text{ and } t = r_j,\dots,T & & (10)
\end{aligned}
$$

It follows from the work of Goemans [Goe96b] that the preemptive schedule in Step 1 of Algorithm 1 is an optimum solution to the relaxation (LP_1). To be more precise, if job j is being processed in period $(t,t+1]$ in the preemptive schedule, then the corresponding variable y_{jt} is set to 1. Henceforth, C_j^{LP} denotes the corresponding "LP completion time" of job j (as defined by (8)). Note that C_j^{LP} is in general *not* the completion time of j in the preemptive schedule!

From the prior discussion, it follows that Step 1 of Algorithm 1 simply computes an optimum solution to the LP relaxation, as does Step 1 of Algorithm R. Moreover, it can easily be seen that for each job j the probability distribution of the random variable t_j in Algorithm R exactly equals the probability distribution of $C_j(\alpha_j)$, if α_j is uniformly distributed in $(0,1]$. For this, observe that the probability that $C_j(\alpha_j) \in (t,t+1]$ for some t equals the fraction y_{jt}/p_j of job j that is being processed in this time interval. Moreover, since α_j is uniformly distributed in $(0,1]$ each point in $(t,t+1]$ is equally likely to be obtained for $C_j(\alpha_j)$. Therefore, the random choice of $C_j(\alpha_j)$ in Algorithm 1 is an alternative way of choosing t_j as it is done in Algorithm R. Consequently, Algorithm 1 with uniformly distributed α_j's is a reformulation of Algorithm R for the single machine case. In particular, the expected completion time of each job is bounded from above by twice its LP completion time and Algorithm 1 is a 2–approximation algorithm.

At this point, it might be appropriate to briefly compare our single machine result with that of Goemans [Goe97]. In Step 2, if we only work with one α for all jobs instead of individual and independent α_j's and if we draw α uniformly from $(0,1]$, then we precisely get Goemans' randomized 2–approximation algorithm RANDOM_α [Goe97]. In particular, Algorithm 1 has the same running time $O(n \log n)$ as RANDOM_α. However, whereas Goemans' algorithm has a small sample space — the algorithm can only output $O(n)$ different schedules — Algorithm 1 can construct exponentially many different schedules.

The following corollary of Lemma 2.1 a) highlights the relation between the LP completion time and the completion time of the preemptive schedule which is optimum for the LP. This observation in the single

machine case is due to Goemans [Goe97]. We will use this crucial insight in the proof of the forthcoming Theorem 3.2.

Corollary 3.1. *Consider any job $j \in J$ and assume that α_j is uniformly distributed between 0 and 1. Then, the expected value $\int_0^1 C_j(\alpha_j)\ d\alpha_j$ of $C_j(\alpha_j)$ in Algorithm 1 is $\frac{1}{p_j}\sum_{t=0}^{T} y_{jt}(t+\frac{1}{2})$. Here, y represents the preemptive schedule constructed in Step 1 of Algorithm 1.*

The second ingredient on our way to a better performance guarantee (besides the use of individual α_j's) is a more intricate density function which is optimal for our analysis. It somewhat reflects a way to deal with the tradeoff between the undesired idle time caused by a job and the just as undesired delay of successive jobs.

Theorem 3.2. *Let each α_j be chosen from a probability distribution over $(0,1]$ with the density function $f(x) = \gamma e^x$ if $x \in (0, x_0 + \ln(2-x_0)]$, and $f(x) = 0$, otherwise, where $x_0 = 0.4835$ and $\gamma = 1/\big((2-x_0)\,e^{x_0} - 1\big)$. Then, the expected completion time of every job $j \in J$ in the schedule constructed by Algorithm 1 is at most $1.6853 \cdot C_j^{LP}$.*

Before the proof of Theorem 3.2 we start with some observations. First, for each job $j \in J$, notice that the expected value $E(\alpha_j)$ is not larger than 0.51655. Moreover, $1+\gamma \leqslant 1.51655 \cdot f(x_0)$ and the expressions $1.51655 \cdot f(x_0)$ and $2 \cdot \big(1 - 0.51655 \cdot E(\alpha_j)\big)$ are bounded from above by 1.6853. We also need the following lemma whose proof is omitted for reasons of brevity.

Lemma 3.3. *Let f be defined as in Theorem 3.2. Then the following holds:*

a) $\int_0^\beta f(x)\ dx + \int_0^\beta f(x) \cdot x\ dx + \beta \int_\beta^1 f(x)\ dx \leqslant (1+\gamma) \cdot \beta$ *for all* $\beta \in [0,1]$.

b) $\int_0^1 f(\alpha_j) \cdot \big(C_j(\alpha_j) - C_j(0^+)\big)\ d\alpha_j \leqslant f(x_0) \int_0^1 \big(C_j(\alpha_j) - C_j(0^+)\big)\ d\alpha_j$, *for all $j \in J$, where $C_j(0^+)$ denotes the starting time and C_j the completion time of job j in the preemptive schedule, respectively.*

Furthermore, we slightly modify the last part of Algorithm 1, i. e., we replace Step 3 by 3' as formulated below. Just for the purpose of a more accessible analysis we convert the preemptive schedule we started with step by step into a nonpreemptive one. We therefore consider the jobs in nonincreasing order of $C_j(\alpha_j)$. A very similar procedure is used in [CMNS97].

3') Take the preemptive schedule we started with in the first step of Algorithm 1. Consider the jobs $j \in J$ in nonincreasing order of $C_j(\alpha_j)$ and iteratively change the current preemptive schedule by applying the following steps:
   - i) remove the $\alpha_j \cdot p_j$ units of job j that are processed before $C_j(\alpha_j)$ from the machine and leave it idle within the corresponding time intervals; we say that this idle time is caused by job j;
   - ii) postpone the whole processing that is done later than $C_j(\alpha_j)$ by p_j;
   - iii) remove the remaining $(1-\alpha_j)$–fraction of job j from the machine and shrink the corresponding time intervals;
   - iv) process job j in the released time interval $(C_j(\alpha_j), C_j(\alpha_j) + p_j]$.

Appendix B contains an example illustrating the action of Step 3'. At the end of Step 3' we have constructed a nonpreemptive schedule from the preemptive schedule we started with. Since the jobs are again scheduled in nondecreasing order of $C_j(\alpha_j)$, the schedule constructed in Step 3 is at least as good as the new one. Therefore it suffices to analyze this new schedule in the following proof of Theorem 3.2.

Proof of Theorem 3.2. Consider an arbitrary, but fixed job $j \in J$ together with a fixed α_j and a certain output of Algorithm 1. We partition the jobs in $J \setminus \{j\}$ into three subsets with regard to their starting time in the preemptive schedule. Let J_1 be the set of all jobs that are started before j. Let further J_2 denote the set of all jobs that are started later than j but before $C_j(\alpha_j)$. We put the remaining jobs of $J \setminus \{j\}$ into subset J_3.

Notice that for each job $k \in J_2$ we have $\frac{w_k}{p_k} > \frac{w_j}{p_j}$ (or $\frac{w_k}{p_k} = \frac{w_j}{p_j}$ and k has smaller index than j) since j is interrupted by k. Hence k is finished before $C_j(\alpha_j)$ in the preemptive schedule. Notice further that no job $k \in J_1$ is being processed between the starting and the completion time of j in the preemptive schedule. If k is not yet completed when j is started, then $\frac{w_k}{p_k} < \frac{w_j}{p_j}$ (or the ratios are equal and j has smaller index).

Now observe that the completion time of j is exactly the sum of its own processing time p_j, the idle time before its start, and the sum of processing times of jobs that start before j. In order to get a better bound than in the proof of Theorem 2.2, we have to refine our analysis of the idle time before the start of job j. This idle time can be written as the sum of the idle time t_{idle} that already existed in the preemptive schedule before $C_j(\alpha_j)$ plus the idle time caused by j itself in Step 3'i plus the sum of the idle times caused by jobs $k \in J \setminus \{j\}$ before the start of job j. Notice that there is no idle time between the start of job j and $C_j(\alpha_j)$ in the preemptive schedule.

To analyze the expected completion time of job j, we again consider a conditional expectation (under the condition α_j). For $k \in J \setminus \{j\}$ let $\beta_k \in [0,1]$ be the fraction of job k that is completed in the preemptive schedule by time $C_j(\alpha_j)$. Note that $\beta_k = 1$ for $k \in J_2$ and $\beta_k = 0$ for $k \in J_3$. Since α_k is chosen independently from α_j, we can write the conditional expected processing time of any job $k \neq j$ plus the conditional expected idle time caused by it before j is started as

$$p_k \int_0^{\beta_k} f(\alpha_k)\, d\alpha_k + p_k \cdot \Big(\int_0^{\beta_k} f(\alpha_k) \cdot \alpha_k \, d\alpha_k + \beta_k \int_{\beta_k}^1 f(\alpha_k)\, d\alpha_k \Big) \ . \tag{11}$$

This sum equals 0 if $k \in J_3$ since $\beta_k = 0$ for such a job k. If $k \in J_1$ the sum is at most $(1+\gamma) \cdot \beta_k \cdot p_k$ by Lemma 3.3 a). Finally, since $\beta_k = 1$ for $k \in J_2$, in this case the sum is exactly $\big(1 + E(\alpha_k)\big) \cdot p_k$. Hence, still for fixed α_j, we can bound the conditional expected completion time of job j as follows:

$$E_{\alpha_j}(C_j) \leqslant p_j + \alpha_j \cdot p_j + t_{idle} + \sum_{k \in J_1} (1+\gamma) \cdot \beta_k \cdot p_k + \sum_{k \in J_2} \big(1 + E(\alpha_k)\big) \cdot p_k \ .$$

If we denote the starting time of job j in the preemptive schedule by $C_j(0^+)$, we have $C_j(0^+) = t_{idle} + \sum_{k \in J_1} \beta_k \cdot p_k$ and $C_j(\alpha_j) - C_j(0^+) = \alpha_j \cdot p_j + \sum_{k \in J_2} p_k$. This yields

$$E_{\alpha_j}(C_j) \leqslant (1+\alpha_j) \cdot p_j + (1+\gamma) \cdot C_j(0^+) + 1.51655 \cdot \big(C_j(\alpha_j) - C_j(0^+) - \alpha_j \cdot p_j \big) \ .$$

To get rid of the conditional expectation we have to integrate over all possible choices of α_j:

$$\begin{aligned} E(C_j) &= \int_0^1 f(\alpha_j) \cdot E_{\alpha_j}(C_j)\, d\alpha_j \\ &\leqslant \big(1 - 0.51655 \cdot E(\alpha_j)\big) \cdot p_j + (1+\gamma) \cdot C_j(0^+) + 1.51655 \int_0^1 f(\alpha_j) \cdot \big(C_j(\alpha_j) - C_j(0^+) \big)\, d\alpha_j \\ &\leqslant 1.6853 \cdot \frac{p_j}{2} + 1.51655 \cdot f(x_0) \int_0^1 C_j(\alpha_j)\, d\alpha_j \ \leqslant \ 1.6853 \cdot C_j^{LP} \end{aligned}$$

The second but last inequality follows from Lemma 3.3 b) and the remarks made before, and the last inequality from Corollary 3.1. □

We can further say that Algorithm 1 actually produces a schedule that is simultaneously expected to be near-optimal with respect to both the total weighted completion time objective and the maximum completion time objective. Under the assumptions of Theorem 3.2, the expected makespan of the schedule constructed by Algorithm 1 is at most 1.5166 times the optimum makespan. Bicriteria results of similar spirit have also been presented in [CPS+96, CMNS97, Wan96]. Wang additionally also uses independent random parameters α_j.

We conclude this section on single machine scheduling by showing that the new rounding technique also bears on the corresponding preemptive problem. Here, the processing of a job may repeatedly be interrupted and continued at a later point in time. We consider $1|r_j, pmtn|\sum w_jC_j$ which is strongly NP-hard [LLLRK84]. Notice that (LP_1) also is a relaxation for this problem. Consider again the last step of Algorithm 1 as formulated in Step 3'. In the preemptive case, there is no need for causing idle time by removing the α_j–fraction of job j from the machine in Step i. We rather schedule $\alpha_j \cdot p_j$ units of job j. Thus, in Step ii, we have to postpone the whole processing that is done later than $C_j(\alpha_j)$ only by $(1-\alpha_j)\cdot p_j$, because this is the remaining processing time of job j, which is then scheduled in $\big(C_j(\alpha_j), C_j(\alpha_j)+(1-\alpha_j)\cdot p_j\big]$. Figure 2 in Appendix B illustrates the action of the accordingly modified Step 3'.

Theorem 3.4. *Let each α_j be chosen from a probability distribution over $(0,1]$ with the density function $g(x) = \frac{1}{3}\cdot(1-x)^{-2}$ if $x \in (0, \frac{3}{4}]$ and $g(x) = 0$, else. Then, for each job j, its expected completion time in the preemptive schedule constructed by the modified Algorithm 1 is at most $\frac{4}{3}\cdot C_j^{LP}$.*

The proof of Theorem 3.4 is similar to the one of Theorem 3.2, with the difference that expression (11) is replaced by another term that is bounded from above by $\frac{4}{3}\cdot\beta_k\cdot p_k$ for $k \in J_1$, and by p_k for $k \in J_2$. This yields the expected performance guarantee $\frac{4}{3}$.

Furthermore, notice that Algorithm 1 with the modified Step 3' can easily be turned into an on-line algorithm for both the nonpreemptive and the preemptive problem. In particular, the preemptive schedule in Step 1 can be constructed until time t without any knowledge of jobs that are released afterwards.

Theorem 3.2 and Theorem 3.4 imply that (LP_1) is a 1.6853–relaxation for $1|r_j|\sum w_jC_j$ and a $4/3$–relaxation for $1|r_j, pmtn|\sum w_jC_j$, respectively. Queyranne and Wang [QW96] show that relaxation (LP_1) is not better than $e/(e-1)$ for the former problem; we have instances that show that it is not better than $8/7$ for the latter one.

Finally, observe that as a consequence of Remark 2.3 α–points are not really needed! Instead we simply need to assign a job randomly to one of the time intervals in which it is processed in the preemptive schedule that defines the optimum solution to (LP_1).

4 Identical Parallel Machine Scheduling with Release Dates

In this section, we show that the combinatorial single machine algorithm can be extended to the more general setting in which we have m identical parallel machines. Each job must be nonpreemptively processed by one of these machines, and may be assigned to any of them. Already the problem $P2||\sum w_jC_j$ is NP-hard, see [BCS74, LRKB77]. We consider $P|r_j|\sum w_jC_j$.

We now briefly describe the adaptation of Algorithm 1 to this setting. The first two steps essentially remain the same, in particular we are working with a single machine! That is, for those two steps we kind of reduce the identical parallel machine instance to a single machine instance. However, the single machine is assumed to be m times as fast as each of the original m machines, i. e., the (virtual) processing time of any job j on this (virtual) machine is $\frac{p_j}{m}$ (w. l. o. g., we may assume that p_j is a multiple of m). The weight and the release date of job j remain the same. This kind of single machine relaxation has been used before in

[CMNS97]. In the final step of our identical parallel machine algorithm (let's call it Algorithm P), we resume the original setting. In order to construct a nonpreemptive schedule on the m machines we first assign each job randomly (with probability $\frac{1}{m}$) to one of the machines and then, on every machine, we schedule the jobs assigned to this machine as early as possible in nondecreasing order of $C_j(\alpha_j)$, where C_j denotes the completion time of job j in the single machine schedule. In particular, Algorithm P has the same running time $O(n \log n)$ as Algorithm 1.

One crucial insight for the analysis of Algorithm P is that there is again an LP relaxation (say (LP_P)) of the original problem the above preemptive schedule on the fast machine defines an optimum solution of which. The LP is almost defined as (LP_1), we just replace inequality (7) with $\sum_{j \in J} y_{jt} \leqslant m$. Now, if we set $y_{jt} = m$ whenever job j is being processed on the virtual machine in the period $(t, t+1]$ by the preemptive schedule, it again essentially follows from [Goe96b] that y is an optimum solution to (LP_P). Moreover, (LP_P) can be seen as a simplification of (LP_R) for identical parallel machines.

If Algorithm R computes the corresponding optimum solution to (LP_R) in its first step, the random assignment of jobs to machines and points in time is the same as in Algorithm P. Thus Theorem 2.2 implies that the expected completion time of every job $j \in J$ in the schedule constructed by Algorithm P is at most $2 \cdot C_j^{LP}$. Therefore, Algorithm P is a 2–approximation algorithm for $P \,|\, r_j \,|\, \sum w_j C_j$. Notice again that it also works on-line. The analysis implies as well that (LP_P) is a 2–relaxation. In fact, this bound is best possible, for (LP_P). For this, consider an instance with m machines and one job of length m and unit weight. The optimum LP completion time is $\frac{m+1}{2}$, whereas the optimum completion time is m.

Furthermore, since (LP_P) is also a relaxation for $P \,|\, r_j, pmtn \,|\, \sum w_j C_j$ it follows that the (nonpreemptive) schedule constructed by Algorithm P is not worse than twice the optimum preemptive schedule. This improves upon a 3–approximation algorithm due to Hall et al. [HSSW96]. Finally, by a nonuniform choice of the α_j's we are able to improve the algorithms for either variant to give a performance guarantee of slightly less than 2. This improvement, however, depends on m.

The perhaps most appealing aspect of Algorithm P is that the random assignment of jobs to machines does not depend on job characteristics; any job is put with probability $1/m$ to any of the machines. This technique also proves useful for the problem without (non-trivial) release dates. The very same random machine assignment followed by list scheduling in order of nonincreasing ratios w_j/p_j on every machine is a randomized $3/2$–approximation algorithm (see [SS97a] for details). Quite interestingly, its derandomized variant precisely coincides with the WSPT-rule analyzed by Kawaguchi and Kyan [KK86]: list the jobs according to nonincreasing ratios w_j/p_j and schedule the next job whenever a machine becomes available.

Acknowledgements: The authors are grateful to Michel X. Goemans and Yaoguang Wang for stimulating discussions on the single machine case, and to Chandra S. Chekuri, Maurice Queyranne, David B. Shmoys, and Yaoguang Wang for helpful comments on an earlier draft of this paper.

References

[AKP92] B. Awerbuch, S. Kutten, and D. Peleg. Competitive distributed job scheduling. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 571 – 581, 1992.

[Bak74] K. R. Baker. *Introduction to Sequencing and Scheduling*. Wiley, 1974.

[BCS74] J. L. Bruno, E. G. Coffman Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the Association for Computing Machinery*, 17:382 – 387, 1974.

[CJM+96] C. Chekuri, R. Johnson, R. Motwani, B. K. Natarajan, B. R. Rau, and M. Schlansker. Profile–driven instruction level parallel scheduling with applications to super blocks. December 1996. Proceedings of the 29th Annual International Symposium on Microarchitecture (MICRO–29), Paris, France.

[CM96] S. Chakrabarti and S. Muthukrishnan. Resource scheduling for parallel database and scientific applications. June 1996. Proceedings of the 8th ACM Symposium on Parallel Algorithms and Architectures.

[CMNS97] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. In *Proceedings of the 8th ACM–SIAM Symposium on Discrete Algorithms*, pages 609 – 618, 1997.

[CPS+96] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming*, number 1099 in Lecture Notes in Computer Science, pages 646 – 657. Springer, Berlin, 1996. Proceedings of the 23rd International Colloquium (ICALP'96).

[CS97] F. A. Chudak and D. B. Shmoys. Approximation algorithms for precedence–constrained scheduling problems on parallel machines that run at different speeds. In *Proceedings of the 8th ACM–SIAM Symposium on Discrete Algorithms*, pages 581 – 590, 1997.

[DLLX90] X. Deng, H. Liu, J. Long, and B. Xiao. Deterministic load balancing in computer networks. In *Proceedings of the 2nd Annual IEEE Symposium on Parallel and Distributed Processing*, pages 50 – 57, 1990.

[DW90] M. E. Dyer and L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26:255 – 270, 1990.

[GLLRK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287 – 326, 1979.

[Goe96a] M. X. Goemans, November 1996. Personal communication.

[Goe96b] M. X. Goemans. A supermodular relaxation for scheduling with release dates. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, number 1084 in Lecture Notes in Computer Science, pages 288 – 300. Springer, Berlin, 1996. Proceedings of the 5th International IPCO Conference.

[Goe97] M. X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th ACM–SIAM Symposium on Discrete Algorithms*, pages 591 – 598, 1997.

[GQS+97] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang, 1997. In preparation.

[GWW96] M. X. Goemans, J. Wein, and D. Williamson, September 1996. Personal communication.

[HSSW96] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off–line and on–line approximation algorithms. Preprint 516/1996, Department of Mathematics, Technical University of Berlin, Berlin, Germany, 1996. To appear in *Mathematics of Operations Research*.

[HSW96] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off–line and on–line algorithms. In *Proceedings of the 7th ACM–SIAM Symposium on Discrete Algorithms*, pages 142 – 151, 1996.

[KK86] T. Kawaguchi and S. Kyan. Worst case bound of an LRF schedule for the mean weighted flow–time problem. *SIAM Journal on Computing*, 15:1119 – 1129, 1986.

[LLLRK84] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In W. R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 245 – 261. Academic Press, New York, 1984.

[LRKB77] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343 – 362, 1977.

[MNR96] R. Motwani, J. Naor, and P. Raghavan. Randomized approximation algorithms in combinatorial optimization. In D. S. Hochbaum, editor, *Approximation Algorithms for NP–Hard Problems*, chapter 11. Thomson, 1996.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MSS96] R. H. Möhring, M. W. Schäffter, and A. S. Schulz. Scheduling jobs with communication delays: Using infeasible solutions for approximation. In J. Diaz and M. Serna, editors, *Algorithms – ESA'96*, volume 1136 of *Lecture Notes in Computer Science*, pages 76 – 90. Springer, Berlin, 1996. Proceedings of the 4th Annual European Symposium on Algorithms.

[PSW95] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures*, number 955 in Lecture Notes in Computer Science, pages 86 – 97. Springer, Berlin, 1995.

[QW96] M. Queyranne and Y. Wang, November 1996. Personal communication.

[RT87] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365 – 374, 1987.

[Sch96] A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP–based heuristics and lower bounds. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, number 1084 in Lecture Notes in Computer Science, pages 301 – 315. Springer, Berlin, 1996. Proceedings of the 5th International IPCO Conference.

[SS97a] A. S. Schulz and M. Skutella. Random–based scheduling: New approximations and LP lower bounds. Preprint 549/1997, Department of Mathematics, Technical University of Berlin, Berlin, Germany, February 1997. To appear in Springer Lecture Notes in Computer Science, Proceedings of the 1st International Symposium on Randomization and Approximation Techniques in Computer Science (Random'97).

[SS97b] A. S. Schulz and M. Skutella. Scheduling–LPs bear probabilities: Randomized approximations for min–sum criteria. In R. Burkard and G. Woeginger, editors, *Algorithms – ESA'97*, Lecture Notes in Computer Science. Springer, Berlin, 1997. Proceedings of the 5th Annual European Symposium on Algorithms.

[ST93] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461 – 474, 1993.

[Wan96] Y. Wang. Bicriteria job scheduling with release dates. Manuscript, Max–Planck–Institut für Informatik, Saarbrücken, Germany, November 1996.

A Interval-Indexed Formulation

As mentioned earlier, the Algorithm R for scheduling unrelated parallel machines suffers from the exponential number of variables in the corresponding LP relaxation. However, we can overcome this drawback by using new variables which are not associated with exponentially many time intervals of length 1, but rather with a polynomial number of intervals of geometrically increasing size. This idea was earlier introduced by Hall et al. [HSW96].

For a given $\eta > 0$, L is chosen to be the smallest integer such that $(1+\eta)^L \geqslant T+1$. Consequently, L is polynomially bounded in the input size of the considered scheduling problem. Let $I_0 = \big[0,1\big]$ and for $1 \leqslant \ell \leqslant L$ let $I_\ell = \big((1+\eta)^{\ell-1}, (1+\eta)^\ell\big]$. We denote with $|I_\ell|$ the length of the ℓ–th interval, i. e., $|I_\ell| = \eta(1+\eta)^{\ell-1}$ for $1 \leqslant \ell \leqslant L$. To simplify notation we define $(1+\eta)^{\ell-1}$ to be $\frac{1}{2}$ for $\ell = 0$. We introduce variables $y_{ij\ell}$ for $i = 1,\ldots,m$, $j \in J$, and $\ell = 0,\ldots,L$ with the following interpretation: $y_{ij\ell} \cdot |I_\ell|$ is the time job j is processed on machine i within time interval I_ℓ, or, equivalently: $(y_{ij\ell} \cdot |I_\ell|)/p_{ij}$ is the fraction of job j that is being processed on machine i within I_ℓ. Consider the following linear program in these interval-indexed variables:

$$
\begin{aligned}
& & \text{minimize} \quad & \sum_{j\in J} w_j C_j & & \\
(LP_R^\eta) \qquad & & \text{subject to} \quad & \sum_{i=1}^{m} \sum_{\substack{\ell=0 \\ (1+\eta)^\ell > r_{ij}}}^{L} \frac{y_{ij\ell}\cdot|I_\ell|}{p_{ij}} = 1 \quad \text{for all } j \in J & (12)\\
& & & \sum_{j\in J} y_{ij\ell} \leqslant 1 \quad \text{for } i = 1,\ldots,m \text{ and } \ell = 0,\ldots,L & (13)\\
& & & C_j = \sum_{i=1}^{m}\sum_{\ell=0}^{L} \Big(\frac{y_{ij\ell}\cdot|I_\ell|}{p_{ij}} (1+\eta)^{\ell-1} + \tfrac{1}{2}\cdot y_{ij\ell}\cdot|I_\ell| \Big) \quad \text{for all } j \in J & (14)\\
& & & y_{ij\ell} = 0 \quad \text{for } i = 1,\ldots,m,\ j \in J,\ (1+\eta)^\ell \leqslant r_{ij} & (15)\\
& & & y_{ij\ell} \geqslant 0 \quad \text{for } i = 1,\ldots,m,\ j \in J,\ \ell = 0,\ldots,L & (16)
\end{aligned}
$$

Consider a feasible schedule and assign the values to the variables $y_{ij\ell}$ as defined above. This solution is clearly feasible: constraints (12) are satisfied since a job j consumes p_{ij} time units if it is processed on machine i; constraints (13) are satisfied since the total amount of processing on machine i of jobs that are processed within the interval I_ℓ cannot exceed its size. Finally, if job j is continuously being processed between $C_j - p_{hj}$ and C_j on machine h, then the right hand side of equation (14) is a lower bound on the real completion time. Thus, (LP_R^η) is a relaxation of the scheduling problem $\mathrm{R}\,|\,r_{ij}\,|\sum w_j C_j$.

Since (LP_R^η) is of polynomial size, an optimum solution can be computed in polynomial time. The following algorithm is an adaption of Algorithm R to the new LP:

Algorithm R^η

1) Compute an optimum solution y to (LP_R^η).
2) Assign job j to a machine–interval pair (i, I_ℓ) at random with probability $\frac{y_{ij\ell}\cdot|I_\ell|}{p_{ij}}$; draw t_j from the chosen time interval at random with uniform distribution.
3) Schedule on each machine i the jobs that were assigned to it nonpreemptively as early as possible in nondecreasing order of t_j.

Again, all the random assignments need to be performed independently from each other. The following lemma is a reformulation of Lemma 2.1 b) for Algorithm R^η and can be proved analogously.

Lemma A.1. *The expected processing time of each $j \in J$ in the schedule constructed by Algorithm* R^{η} *equals* $\sum_{i=1}^{m} \sum_{\ell=0}^{L} y_{ij\ell} \cdot |I_\ell|$.

Theorem A.2. *The expected completion time of each job j in the schedule constructed by Algorithm* R^{η} *is at most* $2 \cdot (1+\eta) \cdot C_j^{LP}$*, where C_j^{LP} is the LP completion time (defined by* (14)*) of the optimum solution y we started with in Step* 1.

Proof. We argue almost exactly as in the proof of Theorem 2.2, but rather use Lemma A.1 instead of Lemma 2.1 b). We consider an arbitrary, but fixed job $j \in J$. First, we also consider a fixed assignment of j to machine i and time interval I_ℓ. Again, the conditional expectation of j's starting time equals the expected idle time plus the expected processing time on machine i before j is started.

With similar arguments as in the proof of Theorem 2.2, we can bound the sum of the idle time plus the processing time by $2 \cdot (1+\eta) \cdot (1+\eta)^{\ell-1}$. This, together with Lemma A.1 and (14) yields the theorem. □

For any given $\varepsilon > 0$ we can choose $\eta = \varepsilon/2$. Then, Algorithm R^{η} is a $(2+\varepsilon)$–approximation algorithm. Furthermore, (LP_R^{η}) is a $(2+\varepsilon)$–relaxation of the problem $\mathrm{R} \,|\, r_{ij} \,|\, \sum w_j C_j$.

Of course, as suggested by Remark 2.3, t_j need also not be chosen at random in Step 2 of Algorithm R^{η}.

B An Illustrating Example

Consider the following instance of $1|r_j|\sum w_jC_j$ consisting of the job set $\{1,2,3,4\}$ together with fixed α_j:

job j	r_j	p_j	w_j/p_j	α_j
1	0	3	1	5/12
2	1	3	2	1/4
3	3	1	3	3/4
4	6	2	2	1/4

The following figure illustrates the procedure of Step 3' in Algorithm 1. We start with the preemptive schedule and get rid of preemption by converting the jobs one after another in nonincreasing order of $C_j(\alpha_j)$.

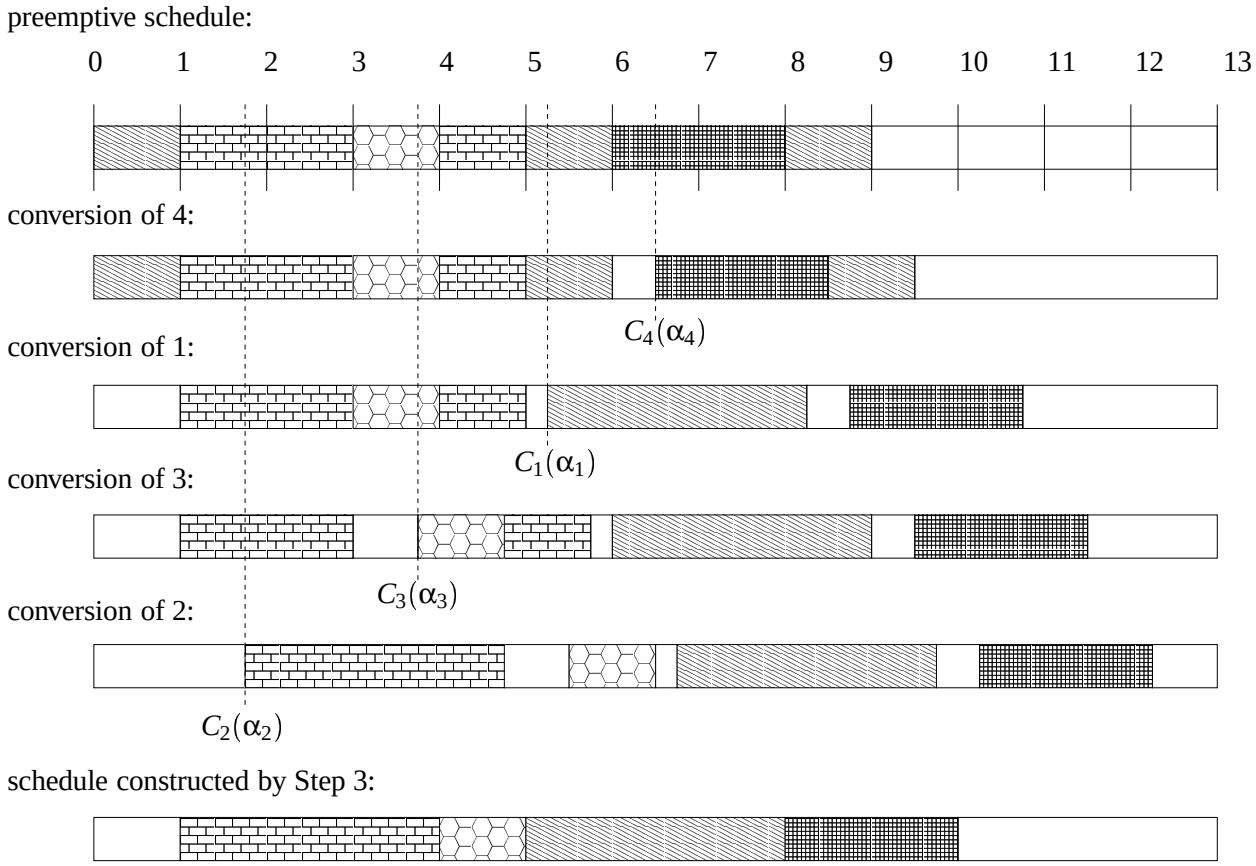

Figure 1: Conversion from preemptive to nonpreemptive schedule in Step 3' resp. 3 of Alg. 1.

Now, we interpret the data considered above as an instance of $1|r_j, pmtn|\sum w_j C_j$. The following figure illustrates the conversion of the preemptive schedule given by the optimum solution to (LP_1) in the modified Algorithm 1:

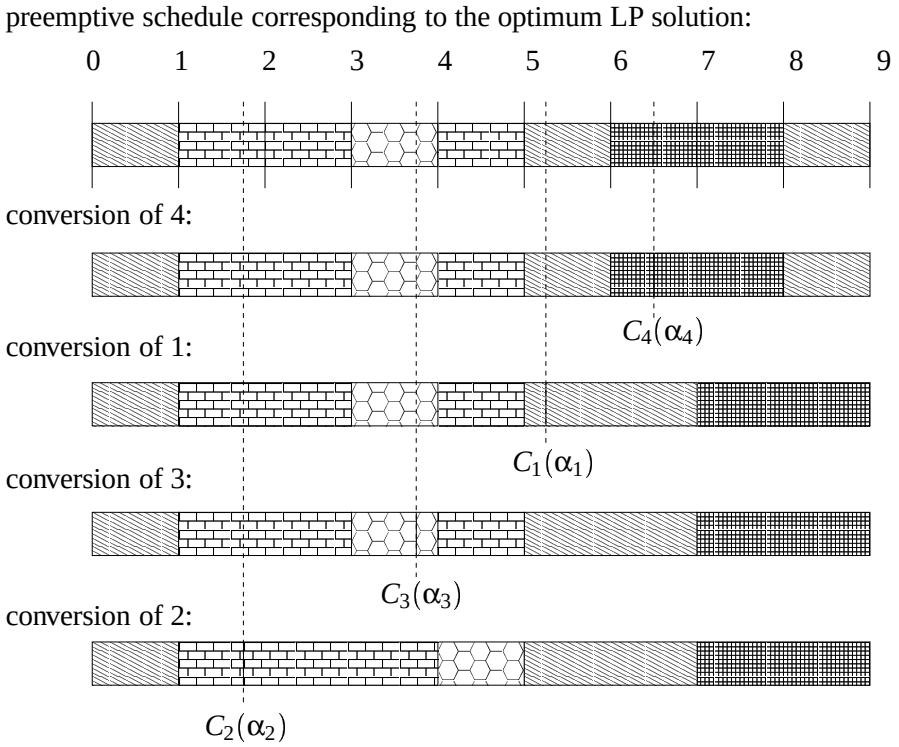

Figure 2: Conversion by the modified Step 3' for the preemptive case.

Reports from the group

"Algorithmic Discrete Mathematics"

of the Department of Mathematics, TU Berlin

554/1997 *Rolf H. Möhring and Matthias Müller–Hannemann:* Complexity and Modeling Aspects of Mesh Refinement into Quadrilaterals

551/1997 *Hans Bodlaender, Jens Gustedt and Jan Arne Telle:* Linear-Time Register Allocation for a Fixed Number of Registers and no Stack Variables

549/1997 *Andreas S. Schulz and Martin Skutella:* Random-Based Scheduling: New Approximations and LP Lower Bounds

536/1996 *Cynthia A. Phillips, Andreas S. Schulz, David B. Shmoys, Cliff Stein, and Joel Wein:* Improved Bounds on Relaxations of a Parallel Machine Scheduling Problem

535/1996 *Rainer Schrader, Andreas S. Schulz, and Georg Wambach:* Base Polytopes of Series-Parallel Posets: Linear Description and Optimization

533/1996 *Andreas S. Schulz and Martin Skutella:* Scheduling–LPs Bear Probabilities: Randomized Approximations for Min–Sum Criteria

530/1996 *Ulrich H. Kortenkamp, Jürgen Richter-Gebert, Aravamuthan Sarangarajan, and Günter M. Ziegler:* Extremal Properties of 0/1-Polytopes

524/1996 *Elias Dahlhaus, Jens Gustedt and Ross McConnell:* Efficient and Practical Modular Decomposition

523/1996 *Jens Gustedt and Christophe Fiorio:* Memory Management for Union-Find Algorithms

520/1996 *Rolf H. Möhring, Matthias Müller-Hannemann, and Karsten Weihe:* Mesh Refinement via Bidirected Flows: Modeling, Complexity, and Computational Results

519/1996 *Matthias Müller-Hannemann and Karsten Weihe:* Minimum Strictly Convex Quadrangulations of Convex Polygons

517/1996 *Rolf H. Möhring, Markus W. Schäffter, and Andreas S. Schulz:* Scheduling Jobs with Communication Delays: Using Infeasible Solutions for Approximation

516/1996 *Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein:* Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms

515/1996 *Christophe Fiorio and Jens Gustedt:* Volume Segmentation of 3-dimensional Images

514/1996 *Martin Skutella:* Approximation Algorithms for the Discrete Time-Cost Tradeoff Problem

509/1996 *Soumen Chakrabarti, Cynthia A. Phillips, Andreas S. Schulz, David B. Shmoys, Cliff Stein, and Joel Wein:* Improved Scheduling Algorithms for Minsum Criteria

508/1996 *Rudolf Müller and Andreas S. Schulz:* Transitive Packing

506/1996 *Rolf H. Möhring and Markus W. Schäffter:* A Simple Approximation Algorithm for Scheduling Forests with Unit Processing Times and Zero-One Communication Delays

505/1996 *Rolf H. Möhring and Dorothea Wagner:* Combinatorial Topics in VLSI Design: An Annotated Bibliography

504/1996 *Uta Wille:* The Role of Synthetic Geometry in Representational Measurement Theory

502/1996 *Nina Amenta and Günter M.Ziegler:* Deformed Products and Maximal Shadows of Polytopes

500/1996 *Stephan Hartmann and Markus W. Schäffter and Andreas S. Schulz:* Switchbox Routing in VLSI Design: Closing the Complexity Gap

498/1996 *Ewa Malesinska, Alessandro Panconesi:* On the Hardness of Allocating Frequencies for Hybrid Networks

496/1996 *Jörg Rambau:* Triangulations of Cyclic Polytopes and higher Bruhat Orders

Reports may be requested from:

S. Marcus
Fachbereich Mathematik, MA 6–1
TU Berlin
Straße des 17. Juni 136
D-10623 Berlin – Germany

e-mail: Marcus@math.TU-Berlin.DE

Reports are available via anonymous ftp from:

ftp.math.tu-berlin.de
cd pub/Preprints/combi
file Report–<number>–<year>.ps.Z